

<http://www.dcs.fmph.uniba.sk/~plachetk/TEACHING/DB1>

Tomáš Plachetka

Fakulta matematiky, fyziky a informatiky,  
Univerzita Komenského, Bratislava

Zima 2024–2025

Príklad:

**vypil(Idn, Alkohol, Mnozstvo)** hovorí, koľko ktorého alkoholu sa vypilo počas návštevy Idn v niektorej krčme

**navstivil(Idn, Pijan, Krcma)** hovorí o návšteve s identifikátorom Idn, ktorého pijana a krčmy sa týka

Dotaz na dvojice [**K, Pocet**], ktoré hovoria, koľko rôznych druhov alkoholu sa vypilo v krčme K

Dotaz na trojice [**K, A, Suma**], ktoré hovoria, aké celkové množstvo alkoholu A sa vypilo v krčme K

Dotaz na dvojice [**K, A**], ktoré hovoria, ktoré alkoholy A sa vypili v krčme K v celkovom množstve aspoň 5

# Grupovanie a agregácia: SQL

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K

```
SELECT n.Krcma, COUNT(DISTINCT v.Alkohol) as Pocet
FROM navstivil n, vypil v
WHERE n.Idn = v.Idn
GROUP BY n.Krcma
```

V krčme slovak sa vypili 2 rôzne druhy alkoholu, nie 3 (preto ten DISTINCT)

navstivil ⋈ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: SQL

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

```
SELECT n.Krcma, v.Alkohol, SUM(v.Mnozstvo) AS Suma
```

```
FROM navstivil n, vypil v
```

```
WHERE n.Idn = v.Idn
```

```
GROUP BY n.Krcma, v.Alkohol
```

Výsledkom je {[slovak, pivo, 6], [slovak, rum, 2]}

navstivil ⋈ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: SQL

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

```
SELECT n.Krcma, v.Alkohol
```

```
FROM navstivil n, vypil v
```

```
WHERE n.Idn = v.Idn
```

```
GROUP BY n.Krcma, v.Alkohol
```

```
HAVING SUM(v.Mnozstvo) >= 5
```

navstivil ⋈ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: relačná algebra

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K

$\Gamma_{Krcma, Pocet} = \text{COUNT}(\text{DISTINCT Alkohol})$  (navstivil  $\bowtie$  vypil)

Alebo, ekvivalentne (všimnite si použitie projekcie  $\pi$  a odstránenia duplikátov  $\delta$ ):

$\Gamma_{Krcma, Pocet} = \text{COUNT}(\text{Alkohol})$  ( $\delta \pi_{Krcma, Alkohol}$  (navstivil  $\bowtie$  vypil))

navstivil $\bowtie$ vypil				
<b>Idn</b>	<b>Alkohol</b>	<b>Mnozstvo</b>	<b>Pijan</b>	<b>Krcma</b>
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: relačná algebra

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

$\Gamma_{\text{Krcma, Alkohol, Suma} = \text{SUM}(\text{Mnozstvo})} (\text{navstivil} \bowtie \text{vypil})$

Aj toto bude fungovať (keďže v relačnej algebre sa duplikáty neodstraňujú automaticky):

$\Gamma_{\text{Krcma, Alkohol, Suma} = \text{SUM}(\text{Mnozstvo})} \pi_{\text{Krcma, Alkohol, Mnozstvo}} (\text{navstivil} \bowtie \text{vypil})$

navstivil $\bowtie$ vypil				
<b>Idn</b>	<b>Alkohol</b>	<b>Mnozstvo</b>	<b>Pijan</b>	<b>Krcma</b>
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: relačná algebra

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

$$\pi_{Krcma, Alkohol} (\sigma_{Suma \geq 5} (\Gamma_{Krcma, Alkohol, Suma = SUM(Mnozstvo)} (navstivil \bowtie vypil)))$$

Výsledkom je {[slovak, pivo]}

navstivil $\bowtie$ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak



# Grupovanie a agregácia: relačný kalkul

Grupovanie a agregácia sa chápe ako rozšírenie všeobecného kvantifikátora. **Agregačný operátor  $\heartsuit$  viaže (t.j. eliminuje) premenné, ktoré sú vymenované za  $\heartsuit$ ; a viaže tiež premenné, ktoré sú agregované niektorou agregačnou funkciou. **Ostatné premenné sa použijú na grupovanie.** Výsledkom je predikát s grupovacími premennými a novými premennými, ktoré vznikli agregáciou**

(Niektorí autori preferujú symbol  $\heartsuit$ , lenže ten nie je dostupný vo fontoch všetkých súčasných počítačových systémov.)

Relačný kalkul pracuje s **množinami** (bez duplikátov), takže prirodzene nepotrebuje **DISTINCT**

# Grupovanie a agregácia: relačný kalkul

Nech  $p$  je predikátová formula s voľnými premennými

$$\mathbf{G} = \{G_1, \dots, G_p\}, \mathbf{A} = \{A_1, \dots, A_q\}, \mathbf{B} = \{B_1, \dots, B_r\},$$

ktorá zodpovedá relácii  $P$  s atribútmi  $\mathbf{G} \cup \mathbf{A} \cup \mathbf{B}$ . Množiny  $\mathbf{G}$ ,  $\mathbf{A}$ ,  $\mathbf{B}$  sú po dvojiciach disjunktné. Nech

$$AGG_i \in \{\text{SUM}, \text{COUNT}, \text{AVG}, \text{MIN}, \text{MAX}\}, i = 1, \dots, q.$$

Potom

$$\heartsuit \mathbf{B}_1, \dots, \mathbf{B}_r, \mathbf{A}_1' = AGG_1(A_1), \mathbf{A}_q' = AGG_q(A_q) (p)$$

je predikátová formula s voľnými premennými

$G_1, \dots, G_p, \mathbf{A}_1', \dots, \mathbf{A}_q'$ , ktorá zodpovedá relácii

$$\Gamma_{\mathbf{G}_1, \dots, \mathbf{G}_p, \mathbf{A}_1' = AGG_1(A_1), \dots, \mathbf{A}_q' = AGG_q(A_q)} (\delta (p))$$

Grupovacie premenné sa v relačnom kalkule píšú „naopak“!

# Grupovanie a agregácia: relačný kalkul

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K

{[K, Pocet]:

♥ Pocet = count(A)

( $\exists I \exists P \exists M$  navstivil(I, P, K)  $\wedge$  vypil(I, A, M))

}

navstivil $\bowtie$ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: relačný kalkul

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

{[K, A, Suma]:

♥ I, Suma = sum(M)

( $\exists P$  navstivil(I, P, K)  $\wedge$  vypil(I, A, M))

}

Premennú I je nutné odprojektovať až počas agregácie, nie skôr! (Prečo?)

navstivil $\bowtie$ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: relačný kalkul

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

{[K, A]:

∃ Suma

♥ I, Suma = sum(M)

(∃ P navstivil(I, P, K) ∧ vypil(I, A, M))

∧ Suma ≥ 5

}

navstivil ⋈ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

Špeciálny predikát **subtotal**, ktorého **prvým argumentom je n-árny predikát  $p$**  s navzájom disjunktnými množinami argumentov  $G_1, \dots, G_p, A_1, \dots, A_q, B_1, \dots, B_r$ . Premenné  $B_1, \dots, B_r$  mimo subtotalu neexistujú

**Druhým argumentom subtotal je zoznam  $[G_1, \dots, G_p]$ .**

**Tretím argumentom subtotal je zoznam**

**$[A_1' = \text{agg}_1(A_1), \dots, A_q' = \text{agg}_q(A_q)]$** , kde  $\text{agg}_i$  je niektorá z agregáčnych funkcií.

Výsledkom je predikát, ktorý zodpovedá formuli

**♥  $B_1, \dots, B_r, A_1' = \text{AGG}_1(A_1), A_q' = \text{AGG}_q(A_q) p(\dots)$**

Relačný kalkul pracuje s **množinami** (bez duplikátov), takže prirodzene nepotrebuje **DISTINCT**

# Grupovanie a agregácia: Datalog

**vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)**

**[K, Pocet]: počet rôznych druhov alkoholu vypitých v krčme K**

**vypite\_alkoholy(K, A) ← navstivil(I, \_, K), vypil(I, A, \_).**

**answer(K, Pocet) ←**

**subtotal(vypite\_alkoholy(K, A), [K], [Pocet = count(A)]).**

<b>navstivil ⋈ vypil</b>				
<b>Idn</b>	<b>Alkohol</b>	<b>Mnozstvo</b>	<b>Pijan</b>	<b>Krcma</b>
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Grupovanie a agregácia: Datalog

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A, Suma]: celkové množstvo alkoholu A vypité v krčme K

ucet(I, K, A, M)  $\leftarrow$  navstivil(I, \_, K), vypil(I, A, M).

answer(K, A, Suma)  $\leftarrow$

subtotal(ucet(\_, K, A, M), [K, A], [Suma = sum(M)]).

Premennú I je nutné odprojektovať až počas agregácie, nie skôr!

navstivil $\bowtie$ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak



# Grupovanie a agregácia: Datalog

vypil(Idn, Alkohol, Mnozstvo), navstivil(Idn, Pijan, Krcma)

[K, A]: alkohol A sa v krčme K vypil v celkovom množstve aspoň 5

ucet(I, K, A, M) ← navstivil(I, \_, K), vypil(I, A, M).

answer(K, A) ←

subtotal(ucet(\_, K, A, M), [K, A], [Suma = sum(M)]),

**Suma >= 5.**

navstivil ⋈ vypil				
Idn	Alkohol	Mnozstvo	Pijan	Krcma
1	pivo	6	laco	slovak
1	rum	1	laco	slovak
2	rum	1	laco	slovak

# Rekapitulácia dotazovacích jazykov

Dotazy sa dajú vyjadriť v 4 jazykoch, ktoré majú pre bezpečné dotazy rovnakú vyjadrovaciu silu (až na spracovanie duplikátov a podobne), t.j. je možné napísať kompilátor z ľubovoľného jazyka do ľubovoľného

- **Relačný kalkul**: bezpečné predikátové formuly
- **Datalog**: bezpečné predikátové formuly v “kanonickej” forme

• „Kanonický“ **SQL**:

SELECT... FROM... WHERE (+NOT EXISTS)...

GROUP BY... HAVING (+NOT EXISTS)...

UNION...

- **Relačná algebra**: výpočty nad reláciami, s operátormi zjednotenia, rozdielu, joinu, (kartézskeho súčinu), selekcie, projekcie, premenovania a agregácie

Takmer... Zatiaľ **nevieme vyjadriť rekurziu v relačnej algebre, t.j. nevieme počítat' rekurzívne dotazy**

Uvažujme nasledujúci Datalogový (bezpečný) program nad databázou  
**navstivil(I, P, K), vypil(I, Alkohol, Mnozstvo)**

**answer(P, K) ← navstivil(I, P, K), vypil(I, rum, \_), not pilvodku(P).**

**answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, \_), not pilvodku(P).**

**pilvodku(P) ← navstivil(I, P, \_), vypil(I, vodka, \_).**

Predikát  $\text{answer}(P, K)$  je splnený pre dvojice  $[P, K]$  také, že pijan  $P$  pri niektorej návšteve krčmy  $K$  vypil rum alebo pivo, a zároveň  $P$  nikdy (pri žiadnej návšteve ktorejkoľvek krčmy) nevypil vodku.

**Graf závislostí IDB predikátov** (predikátov definovaných v programe):

**answer**  $\overleftarrow{\square}$  **pilvodku**

# Datalog → relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma),  
vypil(Idn, Alkohol, Mnozstvo).

**answer(P, K) ← navstivil(I, P, K), vypil(I, rum, \_), not pilvodku(P).**

**answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, \_), not pilvodku(P).**

**pilvodku(P) ← navstivil(I, P, \_), vypil(I, vodka, \_).**

**answer  $\overleftarrow{\neg}$  pilvodku**

Preklad začína s pilvodku, lebo pilvodku nezávisí od iných IDB predikátov, t.j. v grafe závislostí predikátov do vrcholu pilvodku nevedie žiadna hrana. Všetky pozitívne relačné podciele sa spoja joinom, väzby argumentov sa vyjadria v podmienke joinu. Napokon sa výsledok projekciou prispôsobí hlave pravidla:

**pilvodku :=  $\pi_{n.Pijan} (\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='vodka'} \rho_v(\text{vypil}))$**

# Datalog → relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma),  
vypil(Idn, Alkohol, Mnozstvo).

**answer(P, K) ← navstivil(I, P, K), vypil(I, rum, \_), not pilvodku(P).**

**answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, \_), not pilvodku(P).**

**pilvodku(P) ← navstivil(I, P, \_), vypil(I, vodka, \_).**

Predikát answer je definovaný viacerými pravidlami. Po unifikácii hláv pravidiel (ak hlavy nie sú rovnaké, vieme ich urobiť rovnaké premenovaním premenných) sa každé pravidlo preloží zvlášť a výsledky sa zjednotia do relácie answer:

**answer := (preklad 1. pravidla) ∪ (preklad 2. pravidla)**

# Datalog → relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma), vypil(Idn, Alkohol, Mnozstvo).

**answer(P, K) ← navstivil(I, P, K), vypil(I, rum, \_), not pilvodku(P).**

**answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, \_), not pilvodku(P).**

**pilvodku(P) ← navstivil(I, P, \_), vypil(I, vodka, \_).**

Pozitívna časť 1. pravidla sa preloží ako join, negácia ako množinový rozdiel.

Aby sa rozdiel dal urobiť, treba z toho joinu projekciou nechať len atribút Pijan:

$\pi_{n.Pijan} (\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(\text{vypil})) - \text{pilvodku}$

Lenže vo výsledku chceme dvojice [Pijan, Krcma], a atribút Krcma sme tou projekciou stratili. Vieme ho však dostať späť opätovným (prirodzeným) prijoinovaním pozitívnej časti pravidla:

$\pi_{n.Pijan, n.Krcma} ($

$(\pi_{n.Pijan} (\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(\text{vypil})) - \text{pilvodku}) \bowtie$

$(\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(\text{vypil})))$

# Datalog → relačná algebra

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(I, Pijan, Krcma),  
vypil(I, Alkohol, Mnozstvo).

**answer(P, K) ← navstivil(I, P, K), vypil(I, rum, \_), not pilvodku(P).**

**answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, \_), not pilvodku(P).**

**pilvodku(P) ← navstivil(I, P, \_), vypil(I, vodka, \_).**

Plán výpočtu programu v relačnej algebre (počíta správny výsledok, hoci možno nie efektívne):

**pilvodku :=  $\pi_{n.Pijan} (\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='vodka'} \rho_v(\text{vypil}))$**

**answer :=  $\pi_{n.Pijan, n.Krcma} ($**   
 **$(\pi_{n.Pijan} (\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(\text{vypil})) - \text{pilvodku}) \bowtie$**   
 **$(\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'} \rho_v(\text{vypil})))$**   
 **$\cup \pi_{n.Pijan, n.Krcma} ($**   
 **$(\pi_{n.Pijan} (\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='pivo'} \rho_v(\text{vypil})) - \text{pilvodku}) \bowtie$**   
 **$\rho_n(\text{navstivil}))$**

# Datalog → relačná algebra: antijoin

lubi(Pijan, Alkohol), capuje(Krcma, Alkohol, Cena), navstivil(Idn, Pijan, Krcma),  
vypil(Idn, Alkohol, Mnozstvo).

**answer(P, K) ← navstivil(I, P, K), vypil(I, rum, \_), not pilvodku(P).**

**answer(P, K) ← navstivil(I, P, K), vypil(I, pivo, \_), not pilvodku(P).**

**pilvodku(P) ← navstivil(I, P, \_), vypil(I, vodka, \_).**

Trik s prijoinovaním atribútov “stratených” pri výpočte rozdielu relácií je častý. Operátor rozdielu množín ( $-$ ) sa dá nahradiť všeobecnejším operátorom  $\triangleright$ , ktorému sa hovorí **antijoin** (vo výsledku antijoinu sú všetky záznamy z ľavej relácie, ktoré sa nejoinujú so žiadnym záznamom pravej relácie):

**pilvodku :=  $\pi_{n.Pijan}$  ( $\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='vodka'}$   $\rho_v(\text{vypil})$ )**

**answer :=  $\pi_{n.Pijan, n.Krcma}$  (**

**( $\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='rum'}$   $\rho_v(\text{vypil})$ )  $\triangleright$  pilvodku))**

**$\cup \pi_{n.Pijan, n.Krcma}$  (**

**( $\rho_n(\text{navstivil}) \bowtie_{n.Idn=v.Idn \wedge v.Alkohol='pivo'}$   $\rho_v(\text{vypil})$ )  $\triangleright$  pilvodku))**



**Ak je graf závislostí predikátov acyklický**, tak vieme nájsť také usporiadanie, že každý predikát závisí len od EDB a od predošlých predikátov. V tomto poradí sa budú počítat' relácie prislúchajúce tým predikátom v relačnej algebre

- Ak je viacej pravidiel, ktoré definujú jeden predikát, vypočítaj každé zvlášť a výsledky spoj zjednotením  $\cup$
- Pozitívnu časť tela pravidla prelož ako join  $\bowtie$ , negatívnu ako antijoin  $\triangleright$ . Výsledok prispôsob hlave pravidla vhodnou projekciou

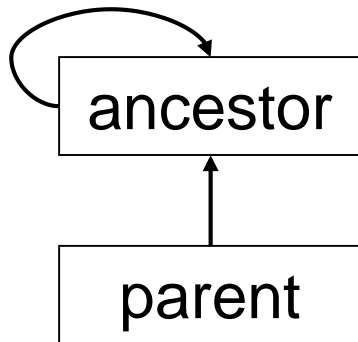
„Malichernosti“ s premenovaním relácií, konštantami, väzbami medzi atribútmi atď. sa vyriešia podobne ako pri preklade

Datalog → SQL

Príklad (predkovia):

```
ancestor(X, A) :- parent(X, A).
```

```
ancestor(X, A) :- parent(X, P), ancestor(P, A).
```



**Rekurzívne programy obsahujú cyklus v grafe závislostí predikátov (hrana z uzla p do uzla q, ak existuje pravidlo s hlavou q, ktorého telo obsahuje p)**

Komplikovanejšia rekurzia (vyjadruje to isté čo predošlý program):

```
ancestor(X, A) :- parent(X, A).
```

```
ancestor(X, A) :- ancestor(X, Y), ancestor(Y, A).
```

## Operátor **fixpoint** (tranzitívny uzáver): $\varphi$

- $\varphi(\langle \text{postupnosť priradení} \rangle)$  iteruje priradenia (každé raz), kým sa relácie v tej postupnosti priradení menia (t.j. iterácia pokračuje ak sa zmení obsah aspoň 1 relácie)

Príklad:

$\text{ancestor} := \text{parent}(X, A)$

$\text{ancestor} := \text{ancestor} \cup \pi_{X, A}(\text{ancestor} \bowtie \text{parent})$

$\text{ancestor} := \text{ancestor} \cup \pi_{X, A}(\text{ancestor} \bowtie \text{parent})$

...

sa zapíše ako

**$\text{ancestor} := \text{parent}$**

**$\varphi(\text{ancestor} := \text{ancestor} \cup \pi_{X, A}(\text{ancestor} \bowtie \text{parent}))$**

Alebo, ekvivalentne (takýto preklad pravidiel funguje všeobecne)

**$\text{ancestor} := \{\}$**

**$\varphi(\text{ancestor} := \text{parent} \cup \pi_{X, A}(\text{ancestor} \bowtie \text{parent}))$**

V relačnom kalkule netreba zavádzať špeciálny operátor na vyjadrenie rekurzie. Rekurzia sa vyjadrí prirodzeným spôsobom

Príklad (predkovia Johna):

ancestor(X, A) :- parent(X, A).

ancestor(X, A) :- parent(X, P), ancestor(P, A).

answer(A) :- ancestor(john, A).

$\{A: \text{ancestor}(\text{john}, A) \wedge$   
 $(\forall X \forall A (\text{parent}(X, A) \Rightarrow \text{ancestor}(X, A))) \wedge$   
 $(\forall X \forall A \forall P ((\text{parent}(X, A) \wedge \text{ancestor}(P, A)) \Rightarrow \text{ancestor}(X, A))\}$

V SQL-99 bola pridaná rekurzia: rekurzívne VIEW, resp. rekurzívne WITH.

Bohužiaľ, niektoré systémy neimplementujú WITH RECURSIVE.

Spoliehajú na cykly resp. rekurziu v externých jazykoch mimo SQL (napr. PHP).

Existujúce systémy, ktoré WITH RECURSIVE implementujú (Oracle, Firebird, DB2), neimplementujú plnú sémantiku (implementujú len rekurziu cez 1 reláciu). Do PostgreSQL bol WITH RECURSIVE dodaný v 2011.

Príklad:

```
WITH RECURSIVE ancestor AS (  
  SELECT p.X FROM parent p  
  UNION  
  SELECT p.X FROM parent p, ancestor a WHERE p.Y=a.X  
)  
SELECT a.X FROM ancestor a
```

# Výpočet bezpečných Datalogových dotazov

Každému (IDB) predikátu  $p_i$  v programe priradiť reláciu  $P_i$

## Algoritmus výpočtu výsledku dotazu (**naivná evaluácia**):

1. Pre všetky  $p_i$  polož  $P_i := \emptyset$
2. do  
{  
    aplikuj výpočet pravidiel každého predikátu  $p_i$  práve raz, v ľubovoľnom poradí (t.j. pre každý predikát vypočítaj zjednotenie joinov/antijoinov s použitím aktuálneho obsahu relácií v tele pravidiel), výsledok priradiť relácii  $P_i$   
} while (niektorá relácia  $P_i$  sa zmenila)
3. Aplikuj výslednú selekciu a projekciu danú dotazom

V kroku 2 sa počíta “výsledok celého programu“ a **až v kroku 3 sa tento výsledok uložený v reláciách  $P_i$  zredukuje na výsledok daného dotazu** (ktorý sa týka len jednej z týchto relácií). Toto môže byť neefektívne, ale funguje korektne pre ľubovoľný program

**Problém: krok 2 v naivnej evaluácii nemusí vždy skončiť**

Príklad:

man(barber). man(mayor).

shaves(barber, X)  $\leftarrow$  man(X), not shaves(X, X).

Počítajme naivnou evaluáciou (relácia man je EDB, tá sa nemení). Predikátu shaves(., .) priradíme reláciu Shaves(X, Y). Iterujeme pravidlo pre Shaves (jediné)

**Shaves := Man**  $\triangleright$   $\text{Man.X} = \text{Shaves.X} \wedge \text{Man.X} = \text{Shaves.Y}$  **Shaves**

Shaves :=  $\emptyset$

Shaves := {[barber, barber], [barber, mayor]}

Shaves := {[barber, mayor]}

Shaves := {[barber, barber], [barber, mayor]}

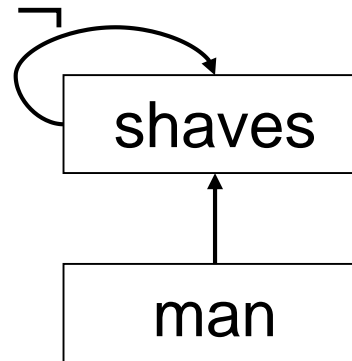
Shaves := {[barber, mayor]}

...

## Problém: krok 2 v naivnej evaluácii nemusí vždy skončiť

?- shaves(barber, mayor) neskončí, ak sa počíta naivnou evaluáciou

Graf závislostí predikátov:



**Pre istú podtriedu dotazov je garantované, že výpočet naivnou evaluáciou skončí**



# Výpočet pevného bodu: Tarskeho veta

**Tarskeho veta o pevnom bode pre úplné zväzy:** ak  $f$  je neklesajúce zobrazenie nad úplným zväzom (t.j.  $x \leq y \Rightarrow f(x) \leq f(y)$ ), tak potom  $f$  má aspoň jeden **pevný bod**  $p$ , pre ktorý platí  $f(p) = p$

Z Tarskeho vety o pevnom bode vyplýva, že **iteratívny výpočet konverguje k pevnému bodu pre “obvyklé” (aj rekurzívne) dotazy**, keďže kartézsky súčin, join, zjednotenie, projekcia a premenovanie sú „neklesajúce“ operácie a extenzionálne relácie sú konečné. **K „neklesajúcim“ operáciám patrí aj stratifikovaná negácia**

**Problém je s nestratifikovanou negáciou (ba aj s bežnou aritmetikou)**

# Stratifikovaná negácia

**Definícia:** program je **stratifikovaný**, keď existuje priradenie  $S(\cdot)$ , ktoré každému predikátu priradí celé číslo, tzv. stratum (vrstva) tak, že platí:

- Ak  $p$  je definovaný pomocou  $q$ , tak  $S(p) \geq S(q)$
- Ak  $p$  je definovaný pomocou  $\neg q$ , tak  $S(p) > S(q)$

Platí: ak je program stratifikovaný a obsahuje  $n$  predikátov, tak tým predikátom možno priradiť strata od 1 po  $n$

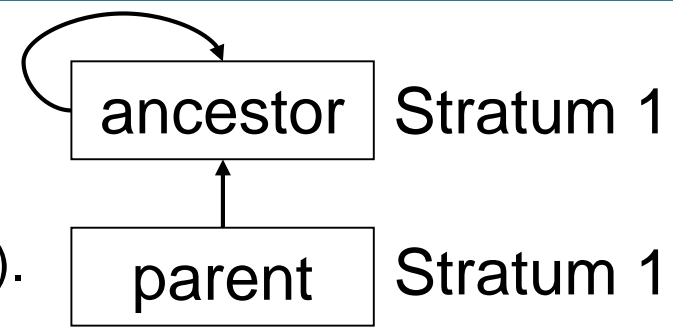
Algoritmus stratifikácie:

1. Všetkým predikátom priradiť stratum 1
2. Ak niektoré pravidlo porušuje podmienku stratifikácie, tak zvýš stratum predikátu v hlave toho pravidla na najmenšie celé číslo, ktoré tú podmienku spĺňa
3. Ak je niektoré stratum väčšie ako počet predikátov, tak program sa nedá stratifikovať

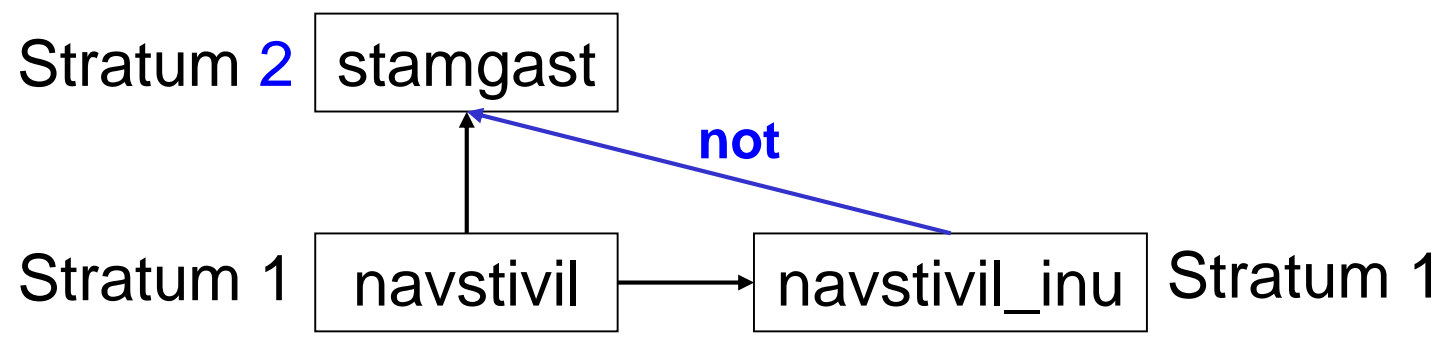
# Stratifikovaná negácia

Príklady:

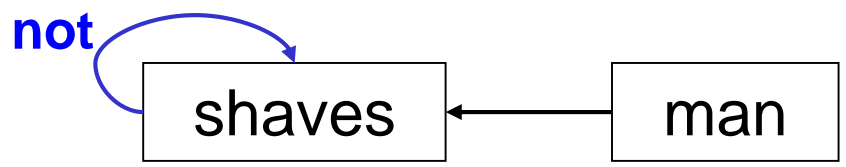
ancestor(X, A) :- parent(X, A).  
ancestor(X, A) :- parent(X, P), ancestor(P, A).



stamgast(P) :- navstivil(\_, P, K), **not** navstivil\_inu(P, K).  
navstivil\_inu(P, K) :- navstivil(\_, P, K), navstivil(\_, P, K2), K ≠ K2.



shaves(barber, X) ← man(X), **not** shaves(X, X).



**Nedá sa stratifikovať**

# Výpočet stratifikovaných Datalogových dotazov

Každému (IDB) predikátu  $s_i$  v programe priradiť reláciu  $S_i$

**Metóda výpočtu výsledku dotazu (naivná evaluácia):**

Začni s prázdnyimi reláciami  $S_i$

Vypočítaj relácie  $S_i$  priradené predikátom v stratum 1

Vypočítaj relácie  $S_i$  priradené predikátom v stratum 2

Vypočítaj relácie  $S_i$  priradené predikátom v stratum 3

...

Poradie výpočtu  $S_i$  v rámci jedného strata:

- Pri výpočte každého strata preferuj najskôr predikáty, ktoré nezávisia od iných IDB predikátov. Relácie pre tieto predikáty stačí vypočítať len raz
- V prípade rekurzívnej definície iteruj výpočet (súčasne) cez všetky predikáty v cykle grafu závislostí, až kým sa prislúchajúce relácie  $S_i$  prestanú meniť. Tieto relácie sa potom už nikdy nebudú meniť

# Výpočet pevného bodu: príklad (vrstovníci)

Relácia  $p(\text{Child}, \text{Parent})$

Napr.  $p = \{[\mathbf{C}l a u d i a, \mathbf{A} d a m], [\mathbf{D} i e t e r, \mathbf{A} d a m], [\mathbf{D} i e t e r, \mathbf{B} r i t t a], [\mathbf{E} r i k, \mathbf{B} r i t t a], [\mathbf{F} r i d d a, \mathbf{C} l a u d i a], [\mathbf{G} i s e l a, \mathbf{C} l a u d i a], [\mathbf{H} e i n z, \mathbf{D} i e t e r], [\mathbf{I} n g o, \mathbf{D} i e t e r], [\mathbf{I} n g o, \mathbf{E} r i k], [\mathbf{F} r i d d a, \mathbf{E} r i k], [\mathbf{J} \ddot{u} r g e n, \mathbf{F} r i d d a], [\mathbf{J} \ddot{u} r g e n, \mathbf{H} e i n z], [\mathbf{K} l a u s, \mathbf{G} i s e l a], [\mathbf{K} l a u s, \mathbf{I} n g o]\}$

Dvojice ľudí  $X$  a  $Y$ , ktorí sú v rovnakej generácii (**same generation**): buď  $X$  a  $Y$  majú spolu dieťa, alebo ich rodičia sú v rovnakej generácii:

$\mathbf{sg}(X, Y) :- p(\mathbf{C}, X), p(\mathbf{C}, Y).$

$\mathbf{sg}(X, Y) :- p(X, \mathbf{P}X), p(Y, \mathbf{P}Y), \mathbf{sg}(\mathbf{P}X, \mathbf{P}Y).$

Treba nájsť všetky dvojice  $X$  a  $Y$ , pre ktoré platí  $\mathbf{sg}(X, Y)$

# Výpočet pevného bodu: iterácia (naivná evaluácia)

```
sg(X, Y) :- p(C, X), p(C, Y).  
sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).
```

Relačná algebra, fixpoint operátor (premenovania sú vynechané):

```
sg(X, Y) := {};  
φ (sg(X, Y) := δ (πX,Y(p(C, X) ⋈ p(C, Y)) ∪  
πX,Y(p(X, PX) ⋈ sg(PX, PY) ⋈ p(Y, PY))));
```

Relačná algebra, naivná evaluácia:

```
sg(X, Y) := {};  
do /* Tento cyklus reprezentuje operátor φ */  
{  
    old_sg(X, Y) := sg(X, Y);  
    sg(X, Y) := δ (πX,Y(p(C, X) ⋈ p(C, Y)) ∪  
πX,Y(p(X, PX) ⋈ sg(PX, PY) ⋈ p(Y, PY))));  
}  
while old_sg(X, Y) <> sg(X, Y);
```

## Výpočet pevného bodu: iterácia (naivná evaluácia)

$sg(X, Y) :- p(C, X), p(C, Y).$

$sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).$

SQL:

```
WITH RECURSIVE sg AS (  
SELECT p1.Parent AS X, p2.Parent AS Y  
FROM p p1, p p2  
WHERE p1.Child = p2.Child  
UNION  
SELECT p1.Child AS X, P2.Child AS Y  
FROM p p1, p p2, sg s  
WHERE p1.Parent = s.X AND p2.Parent = s.Y)  
  
SELECT DISTINCT s.X, s.Y  
FROM sg s
```

# Výpočet pevného bodu: iterácia (naivná evaluácia)

$sg(X, Y) :- p(C, X), p(C, Y).$

$sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).$

SQL, naivná evaluácia s použitím cyklu mimo SQL:

`sg := {};`

`do /* Tento cyklus je implementovaný v operátore  $\varphi$  */`

`{`

`old_sg := sg;`

`sg :=`

`(SELECT p1.Parent AS X, p2.Parent AS Y`

`FROM p p1, p p2`

`WHERE p1.Child = p2.Child`

`UNION`

`SELECT p1.Child AS X, P2.Child AS Y`

`FROM p p1, p p2, sg s`

`WHERE p1.Parent = s.X AND p2.Parent = s.Y);`

`}`

`while (old_sg <> sg);`



# Výpočet pevného bodu: iterácia (naivná evaluácia)

$sg(X, Y) :- p(C, X), p(C, Y).$

$sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).$

$p = \{[\mathbf{C}ludia, \mathbf{A}dam], [\mathbf{D}ieter, \mathbf{A}dam], [\mathbf{D}ieter, \mathbf{B}ritta], [\mathbf{E}rik, \mathbf{B}ritta],$   
 $[\mathbf{F}ridda, \mathbf{C}ludia], [\mathbf{G}isela, \mathbf{C}ludia], [\mathbf{H}einz, \mathbf{D}ieter], [\mathbf{I}ngo, \mathbf{D}ieter], [\mathbf{I}ngo, \mathbf{E}rik],$   
 $[\mathbf{F}ridda, \mathbf{E}rik], [\mathbf{J}ürgen, \mathbf{F}ridda], [\mathbf{J}ürgen, \mathbf{H}einz], [\mathbf{K}laus, \mathbf{G}isela], [\mathbf{K}laus, \mathbf{I}ngo]\}$

i	sg	sg(X, Y)
1	19	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II
2	29	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK
3	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK, GH, HG, JK, KJ
4	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, GH, HG, HI, IF, IH, JJ, JK, KJ, KK

# Výpočet pevného bodu: iterácia (seminaiivná evaluácia)

Optimalizácia: nepočítať opakovane všetky dvojice v každom kroku. Stačí počítať (a pridávať) **len nové dvojice**

Naivná evaluácia:  $19+29+33+33=114$  vypočítaných dvojíc

Seminaivná evaluácia:  $19+10+4+0=33$  vypočítaných dvojíc

i	sgl	sg(X, Y)	\Delta sgl	\Delta sgl(X, Y)
1	19	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II	19	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II
2	29	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK	10	CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK
3	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK, GH, HG, JK, KJ	4	GH, HG, JK, KJ
4	33	AA, AB, BA, BB, CC, CE, DD, DE, EC, ED, EE, FF, FH, GG, GI, HF, HH, IG, II, CD, DC, FG, FI, GF, HI, IF, IH, JJ, KK, GH, HG, JK, KJ	0	

# Výpočet pevného bodu: iterácia (seminatívna evaluácia)

$sg(X, Y) :- p(C, X), p(C, Y).$

$sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).$

Relačná algebra, seminatívna evaluácia (diferenčná schéma):

$sg(X, Y) := \emptyset;$

$\Delta sg(X, Y) := \emptyset;$

do /\* Tento cyklus reprezentuje operátor  $\varphi$  \*/

{

$\Delta sg(X, Y) := (\pi_{X,Y}(p(C, X) \bowtie p(C, Y)) \cup$   
 $\pi_{X,Y}(p(X, PX) \bowtie p(Y, PY) \bowtie \Delta sg(PX, PY)))$   
 $- sg(X, Y);$

$sg(X, Y) := sg(X, Y) \cup \Delta sg(X, Y);$

}

$while (\Delta sg(X, Y) \neq \emptyset);$

# Výpočet pevného bodu: iterácia (seminatívna evaluácia)

$sg(X, Y) :- p(C, X), p(C, Y).$

$sg(X, Y) :- p(X, PX), p(Y, PY), sg(PX, PY).$

Seminaívna evaluácia, optimalizovaná verzia (prvé pravidlo prispeje do  $\Delta sg$  iba raz):

$sg(X, Y) := \emptyset;$

$\Delta sg(X, Y) := \pi_{X,Y}(p(C, X) \bowtie p(C, Y));$

do /\* Tento cyklus reprezentuje operátor  $\Phi$  \*/

{

$\Delta sg(X, Y) := (\pi_{X,Y}(p(X, PX) \bowtie p(Y, PY) \bowtie \Delta sg(PX, PY)))$   
 $\quad - sg(X, Y);$

$sg(X, Y) := sg(X, Y) \cup \Delta sg(X, Y);$

}

$while (\Delta sg(X, Y) \neq \emptyset);$

# Výpočet pevného bodu: iterácia (seminatívna evaluácia)

Embedded SQL, seminatívna evaluácia (s iteráciou mimo SQL):

```
sg := ∅;  
Δsg := ∅;  
do /* Tento cyklus reprezentuje operátor φ */  
    Δsg :=  
        SELECT newsg.X, newsg.Y  
        FROM  
            (SELECT p1.Parent AS X, p2.Parent AS Y  
             FROM p p1, p p2 J  
             WHERE p1.Child = p2.Child  
             UNION  
             SELECT p1.Child AS X, P2.Child AS Y  
             FROM p p1, p p2, sg s  
             WHERE p1.Parent = s.X AND p2.Parent = s.Y) newsg  
        WHERE [s.X, s.Y] NOT IN  
            (SELECT oldsg.X, oldsg.Y FROM sg oldsg);  
    sg(X, Y) := sg(X, Y) ∪ Δsg(X, Y);  
while (Δsg(X, Y) <> ∅);
```

## Ešte príklad: emp(Id, Name, Mgr), Datalog

emp(Employee\_Id, Employee\_Name, Manager\_Id)

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

Datalog:

answer(N)  $\leftarrow$  emp(Id, N, \_), super\_blake(Id).

super\_blake(Id)  $\leftarrow$  emp(\_, blake, Id).

super\_blake(Id)  $\leftarrow$  emp(Mid, \_, Id), super\_blake(Mid).

Lenže obyčajne sa snažíme programovať všeobecné predikáty (super\_blake je príliš jednoúčelový). Nasledujúca verzia je všeobecnejšia:

**super(Id, Mid)  $\leftarrow$  emp(Id, \_, Mid).**

**super(Id, Mid)  $\leftarrow$  emp(Id, \_, Mid2), super(Mid2, Mid).**

**answer(N)  $\leftarrow$  emp(Mid, Sname, \_), emp(Id, blake, \_), super(Id, Mid).**

Pred výpočtom dotazu „? answer(N)“ dobrý kompilátor/optimalizátor **automaticky** transformuje tento všeobecný program na program, ktorý je veľmi podobný tomu prvému programu

## Ešte príklad: emp(Id, Name, Mgr), relačná algebra

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

$\text{super}(\text{Id}, \text{Mid}) \leftarrow \text{emp}(\text{Id}, \_, \text{Mid}).$

$\text{super}(\text{Id}, \text{Mid}) \leftarrow \text{emp}(\text{Id}, \_, \text{Mid2}), \text{super}(\text{Mid2}, \text{Mid}).$

$\text{answer}(\text{Mname}) \leftarrow \text{emp}(\text{Mid}, \text{Mname}, \_), \text{emp}(\text{Id}, \text{blake}, \_), \text{super}(\text{Id}, \text{Mid}).$

Relačná algebra:

$\varphi$  ( $\text{super} := \pi_{\text{Id}, \text{Mgr}} \text{emp} \cup \pi_{\text{emp.Id}, \text{super.Mgr}} \text{emp} \bowtie_{\text{emp.Mgr} = \text{super.Id}} \text{super}$ ;

$\text{answer} := \sigma_{\text{Name}='blake'} (\rho_{e2} (\text{emp})) \bowtie_{e2.\text{Id} = s.\text{Id}} \rho_s (\text{super}) \bowtie_{e1.\text{Id} = s.\text{Mgr}} \rho_{e1} (\text{emp}))$

Finálny dotaz:

answer

## Ešte príklad: emp(Id, Name, Mgr), relačný kalkul

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

$\text{super}(\text{Id}, \text{Mid}) \leftarrow \text{emp}(\text{Id}, \_, \text{Mid}).$

$\text{super}(\text{Id}, \text{Mid}) \leftarrow \text{emp}(\text{Id}, \_, \text{Mid2}), \text{super}(\text{Mid2}, \text{Mid}).$

$\text{answer}(\text{Mname}) \leftarrow \text{emp}(\text{Mid}, \text{Mname}, \_), \text{emp}(\text{Id}, \text{blake}, \_), \text{super}(\text{Id}, \text{Mid}).$

Relačný kalkul:

{Mname:

$(\exists \text{Mid} \exists X1 \exists X2 \exists \text{Id} \text{emp}(\text{Mid}, \text{Mname}, X1) \wedge \text{emp}(\text{Id}, \text{blake}, X2) \wedge \text{super}(\text{Id}, \text{Mid}))$

$\wedge$

$(\forall \text{Id} \forall \text{Mid} \forall N (\text{emp}(\text{Id}, N, \text{Mid}) \Rightarrow \text{super}(\text{Id}, \text{Mid}))) \wedge$

$(\forall \text{Id} \forall \text{Mid} \forall \text{Mid2} \forall N ((\text{emp}(\text{Id}, N, \text{Mid2}) \wedge \text{super}(\text{Mid2}, \text{Mid})) \Rightarrow \text{super}(\text{Id}, \text{Mid}))))$



# Ešte príklad: emp(Id, Name, Mgr), WITH RECURSIVE

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

```
super(Id, Mid) ← emp(Id, _, Mid).  
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).  
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).
```

SQL s WITH RECURSIVE:

```
with recursive super as  
(select e.Id, e.Mgr  
from emp e)  
union  
(select e.Id, s.Mgr  
from emp e, super s  
where e.Mgr = s.Id),  
select e1.Name  
from emp e1, emp e2, super s  
where e1.Id = s.Mgr and e2.Id = s.Id and e2.Name = 'blake'
```

## Ešte príklad: emp(Id, Name, Mgr), SQL bez WITH

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

$\text{super}(\text{Id}, \text{Mid}) \leftarrow \text{emp}(\text{Id}, \_, \text{Mid}).$   
 $\text{super}(\text{Id}, \text{Mid}) \leftarrow \text{emp}(\text{Id}, \_, \text{Mid2}), \text{super}(\text{Mid2}, \text{Mid}).$   
 $\text{answer}(\text{Mname}) \leftarrow \text{emp}(\text{Mid}, \text{Mname}, \_), \text{emp}(\text{Id}, \text{blake}, \_), \text{super}(\text{Id}, \text{Mid}).$

SQL bez WITH:

```
create temporary table super1 as
select e.Id, e.Mgr
from emp e;
```

```
create temporary table super2 as
((select e.Id, e.Mgr
from emp e)
union
(select e.Id, s.Mgr
from emp e, super1 s
where e.Mgr = s.Id));
```

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

```
super(Id, Mid) ← emp(Id, _, Mid).  
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).  
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).
```

SQL bez WITH, pokračovanie:

...

```
create temporary table super10 as  
((select e.Id, e.Mgr  
from emp e)  
union  
(select e.Id, s.Mgr  
from emp e, super9 s  
where e.Mgr = s.Id));
```

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

```
super(Id, Mid) ← emp(Id, _, Mid).  
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).  
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).
```

SQL bez WITH, finálny SELECT (pri hĺbke rekurzie 10):

```
select e1.Name  
from emp e1, emp e2, super10 s  
where e1.Id = s.Mgr and e2.Name = 'blake' and e2.Id = s.Id;
```

Pre ľubovoľnú fixnú hĺbku rekurzie je možné rekurziu simulovať aj bez použitia WITH RECURSIVE (musí byť známe horné ohraničenie tej hĺbky)

Vyskúšajte na databáze emp (stačí hĺbka rekurzie 3, netreba 10)

**Nájdite všetkých nadriadených zamestnanca BLAKE (priamych aj nepriamych).**

```
super(Id, Mid) ← emp(Id, _, Mid).  
super(Id, Mid) ← emp(Id, _, Mid2), super(Mid2, Mid).  
answer(Mname) ← emp(Mid, Mname, _), emp(Id, blake, _), super(Id, Mid).
```

SQL bez WITH, seminainvá evaluácia (efektívnejšia ako predošlá naivná):

```
create temporary table super2 as  
(select e.Id, e.Mgr  
from emp e)  
union  
(select e.Id, s.Mgr  
from emp e, super1 s  
where e.Mgr = s.Id and not exists (  
  select *  
  from super1 s1  
  where s1.Id = e.Id and s1.Mgr = s.Mgr));
```

... podobne pre super3 až super10